Circuits combinatoires

Christophe Mouilleron

ensiie
école nationale supérieure d'informatique
pour l'industrie et l'entreprise

Information

Un bit = deux états possibles

- haut = 1
- bas = 0

$\rightsquigarrow$ représentation binaire

différence de 1 V entre les deux états

Information

Un bit = deux états possibles

- haut = 1
- bas = 0

$\rightsquigarrow$ représentation binaire

différence de 1 V entre les deux états

Algèbre de Boole $\mathbb{B}$

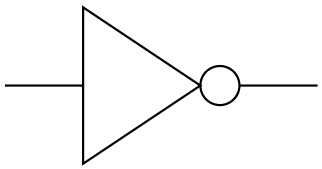
- *true* = $\top$ = 1
- *false* = $\bot$ = 0

$\rightsquigarrow$ correspondance

Portes logiques

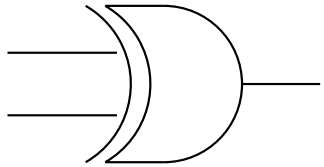
À partir de transistors, on sait faire (entre autres) :

Not



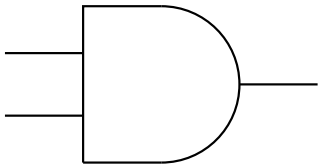
a	$\bar{a}$
0	1
1	0

Xor



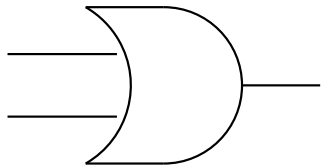
a	b	$a \oplus b$
0	0	0
0	1	1
1	0	1
1	1	0

And



a	b	$a \cdot b$
0	0	0
0	1	0
1	0	0
1	1	1

Or



a	b	$a + b$
0	0	0
0	1	1
1	0	1
1	1	1

Fonctions booléennes

Donnée :

$$f : \mathbb{B}^m \to \mathbb{B}$$

$\rightsquigarrow$ table de vérité

Objectif : créer un circuit calculant f

- fils + portes logiques
- m entrées
- 1 sortie

Fonctions booléennes

Donnée :

$$f : \mathbb{B}^m \to \mathbb{B}^n$$

$\rightsquigarrow$ table de vérité

Objectif : créer un circuit calculant f

- fils + portes logiques
- m entrées
- n sorties

Applications :

- opérations élémentaires
- tables

Newton

Plan

Plan

Table de vérité $\rightarrow$ Circuit

Idée :

- dresser la table de vérité de f
- tester chaque ligne où $f(b_1, \ldots, b_n) = 1$
  $\rightsquigarrow$ portes `Not` et `And` sortie = 1 ssi ligne concernée
- faire un `Ou` de toutes ces lignes

Table de vérité $\rightarrow$ Circuit

Idée :

- dresser la table de vérité de f
- tester chaque ligne où $f(b_1, \ldots, b_n) = 1$
  $\rightsquigarrow$ portes `Not` et `And` sortie = 1 ssi ligne concernée
- faire un `Ou` de toutes ces lignes

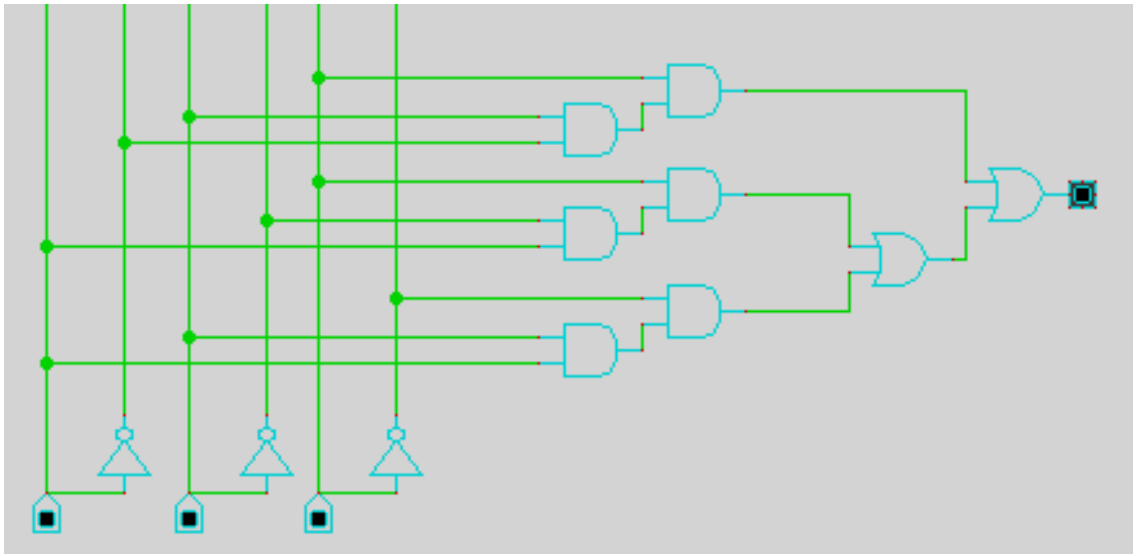
Exemple : $f(b_1, b_2, b_3) = 1$ ssi 2 entrées à 1.

Table de vérité $\rightarrow$ Circuit

Idée :

- dresser la table de vérité de f
- tester chaque ligne où $f(b_1, \ldots, b_n) = 1$
  $\rightsquigarrow$ portes `Not` et `And` sortie = 1 ssi ligne concernée
- faire un `Ou` de toutes ces lignes

Exemple : $f(b_1, b_2, b_3) = 1$ ssi 2 entrées à 1. $\rightsquigarrow$ cf 2among3.lgf



Formes normales

Autres approches =

- simplifier via loi usuelles sur $\mathbb{B}$
- utiliser une forme normale
  - conjonctive
  - disjonctive

$\bigwedge \bigvee$ littéraux
$\bigvee \bigwedge$ littéraux

Formes normales

Autres approches =

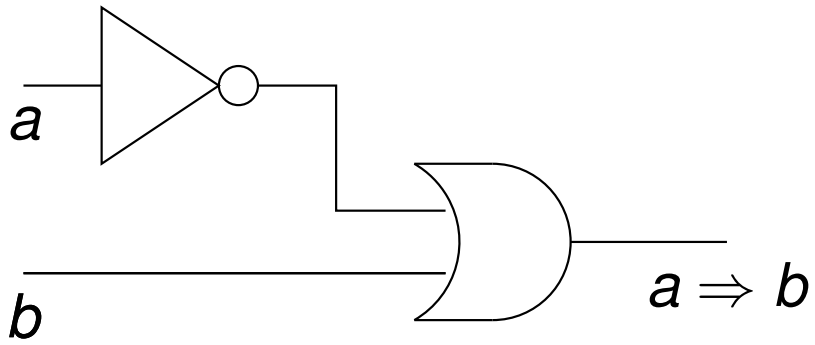
- simplifier via loi usuelles sur $\mathbb{B}$
- utiliser une forme normale
  - conjonctive
  - disjonctive

$\bigwedge \bigvee$ littéraux
$\bigvee \bigwedge$ littéraux

Exemple : $a \Rightarrow b$

Forme normale disjonctive = $b \vee \bar{a}$

Circuit =



Plan

Code de Gray

Idée :

- lister les entiers de 0 à $2^n - 1$ écriture binaire
- passage au suivant en changeant exactement un bit

Exemple : $n = 3$

décimal	binaire	gray
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

Code de Gray, calcul

Méthode math. : récurrence

$n = 1$

0
1

miroir + extension $\longrightarrow$

$n = 2$

0 0
0 1
---
1 1
1 0

miroir + extension $\longrightarrow$

Code de Gray, calcul

Méthode math. : récurrence

$n = 1$

0
1

miroir + extension $\longrightarrow$

$n = 2$

0	0
0	1
1	1
1	0

miroir + extension $\longrightarrow$

$n = 3$

0	00
0	01
0	11
0	10
1	10
1	11
1	01
1	00

Code de Gray, calcul

Méthode math. : récurrence

$n = 1$

0
1

miroir + extension $\longrightarrow$

$n = 2$

0	0
0	1
1	1
1	0

miroir + extension $\longrightarrow$

$n = 3$

0	00
0	01
0	11
0	10
1	10
1	11
1	01
1	00

Méthode `C` : `i ^ (i>>1)`

$n = 4$, $i = 6$?

$$\begin{array}{cccc} & 0 & 1 & 1 & 0 \\ \oplus & 0 & 0 & 1 & 1 \\ \hline & 0 & 1 & 0 & 1 \end{array}$$

Table de vérité $\rightarrow$ Table(au) de Karnaugh

Idée : réorganiser la table de vérité en utilisant le code de Gray

a	*b*	*c*	*d*	out
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Table de Karnaugh

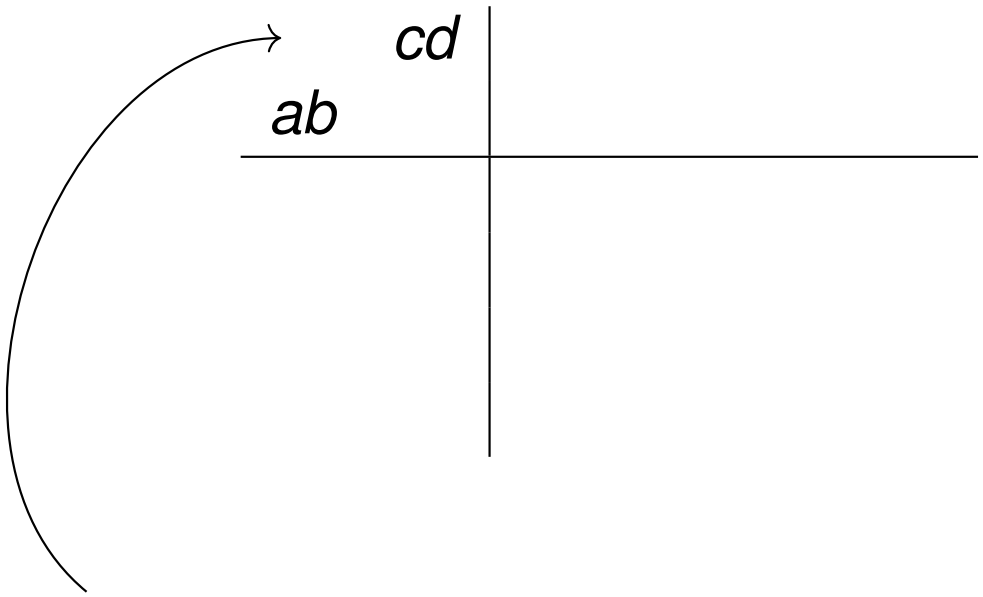
ab \ *cd*				

Table de vérité → Table(au) de Karnaugh

Idée : réorganiser la table de vérité en utilisant le code de Gray

a	*b*	*c*	*d*	out
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Table de Karnaugh



séparer les entrées
en deux groupes

Table de vérité → Table(au) de Karnaugh

Idée : réorganiser la table de vérité en utilisant le code de Gray

a	*b*	*c*	*d*	out
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Table de Karnaugh

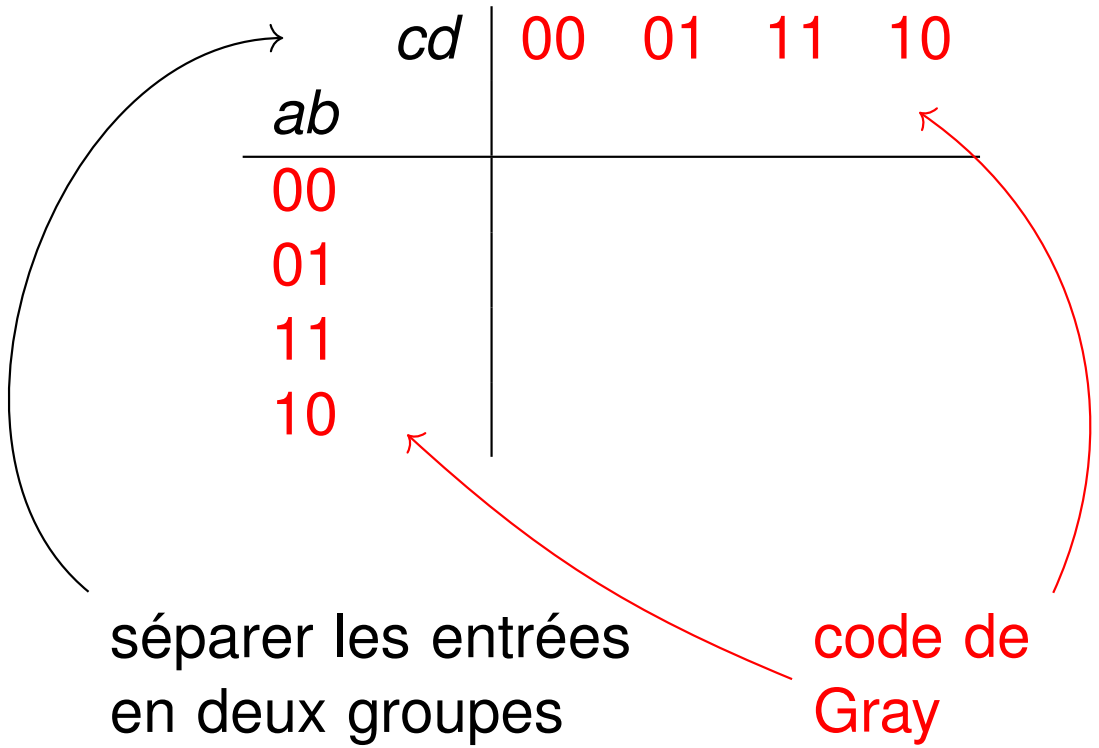


Table de vérité → Table(au) de Karnaugh

Idée : réorganiser la table de vérité en utilisant le code de Gray

a	*b*	*c*	*d*	out
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Table de Karnaugh

ab \ *cd*	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	1	1	1	1
10	1	0	0	1

séparer les entrées en deux groupes

code de Gray

Table de Karnaugh $\rightarrow$ Circuit

ab \ *cd*	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	1	1	1	1
10	1	0	0	1

- 1 ligne = *a* et *b* fixés

Table de Karnaugh → Circuit

ab \ *cd*	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	1	1	1	1
10	1	0	0	1

- 1 ligne = *a* et *b* fixés
- 2 lignes adjacentes = *a* ou *b* fixé

Table de Karnaugh → Circuit

ab \ *cd*	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	1	1	1	1
10	1	0	0	1

- 1 ligne = *a* et *b* fixés
- 2 lignes adjacentes = *a* ou *b* fixé
- 1 colonne = *c* et *d* fixés

Table de Karnaugh $\rightarrow$ Circuit

ab \ *cd*	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	1	1	1	1
10	1	0	0	1

- 1 ligne = a et b fixés
- 2 lignes adjacentes = a ou b fixé
- 1 colonne = c et d fixés
- carré 2×2 = 2 entrées fixées

Table de Karnaugh $\rightarrow$ Circuit

ab \ *cd*	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	1	1	1	1
10	1	0	0	1

- 1 ligne = a et b fixés
- 2 lignes adjacentes = a ou b fixé
- 1 colonne = c et d fixés
- carré $2 \times 2 = 2$ entrées fixées
- ...

Méthode : paver les 1 par des rectangles $2^i \times 2^j$ les + gros possible

Table de Karnaugh $\rightarrow$ Circuit

ab \ cd	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	1	1	1	1
10	1	0	0	1

- 1 ligne = a et b fixés
- 2 lignes adjacentes = a ou b fixé
- 1 colonne = c et d fixés
- carré 2×2 = 2 entrées fixées
- ...

Méthode : paver les 1 par des rectangles $2^i \times 2^j$ les + gros possible

Sur l'exemple :

- $b = 1$

b

Table de Karnaugh $\rightarrow$ Circuit

ab \ cd	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	1	1	1	1
10	1	0	0	1

- 1 ligne = a et b fixés
- 2 lignes adjacentes = a ou b fixé
- 1 colonne = c et d fixés
- carré 2×2 = 2 entrées fixées
- ...

Méthode : paver les 1 par des rectangles $2^i \times 2^j$ les + gros possible

Sur l'exemple :

- $b = 1$ $\qquad b$
- $a = 1$ et $d = 0$ $\qquad a \cdot \bar{d}$

Table de Karnaugh $\rightarrow$ Circuit

ab \ cd	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	1	1	1	1
10	1	0	0	1

- 1 ligne = a et b fixés
- 2 lignes adjacentes = a ou b fixé
- 1 colonne = c et d fixés
- carré 2×2 = 2 entrées fixées
- ...

Méthode : paver les 1 par des rectangles $2^i \times 2^j$ les + gros possible

Sur l'exemple :

- $b = 1$ $\qquad b$
- $a = 1$ et $d = 0$ $\qquad a \cdot \bar{d}$

$\rightsquigarrow a \cdot \bar{d} + b$ $\qquad$ 1 `Not`, 1 `And`, 1 `Or`

Plan

Décodeurs (ou démultiplexeurs)

n entrées : $i_0, i_1, \ldots, i_{n-1}$

$\rightsquigarrow$ entier i codé en binaire $\quad i = \sum_{k=0}^{n-1} i_k \, 2^k$

2^n sorties : $s_0, \ldots, s_{2^n-1}$

- $s_j = 1$ quand $i = j$
- $s_j = 0$ sinon

cf. exemple dans Diglog

Multiplexeurs

$k + n2^k$ entrées :

- 2^k données de n bits
- k bits de contrôle

n sorties :

- donnée numéro i, où i = valeur indiquée dans les bits de contrôle

Multiplexeurs

$k + n2^k$ entrées :

- 2^k données de n bits
- k bits de contrôle

n sorties :

- donnée numéro i, où i = valeur indiquée dans les bits de contrôle

Exemple : Multiplexeur $8 \rightarrow 4$

- $2^k = 8/4 = 2$, $k = 1$

Multiplexeurs

$k + n2^k$ entrées :

- 2^k données de n bits
- k bits de contrôle

n sorties :

- donnée numéro i, où i = valeur indiquée dans les bits de contrôle

Exemple : Multiplexeur $8 \rightarrow 4$

- $2^k = 8/4 = 2$, $k = 1$
- 4 sorties, $8 + 1$ entrées
- choix de 4 fils selon le bit de contrôle test

Complément à 1

n entrées : $i_0, i_1, \ldots, i_{n-1}$

n sorties : $s_0, \ldots, s_{n-1}$

- $s_j = \bar{i_j}$ pour tout $j \in [\![0,\ n-1]\!]$

n portes `Not` en parallèle